Package 'Rmosek'

February 19, 2015

Version 1.2.5.1

Title The R-to-MOSEK Optimization Interface

Author Henrik Alsing Friberg

Maintainer Henrik Alsing Friberg <haf@mosek.com>

Description An interface to the MOSEK optimization library designed to solve large-scale mathematical optimization problems. Supports linear, quadratic and second order cone optimization with/without integer variables, in addition to the more general separable convex problems. Trial and free academic licenses available at http://www.mosek.com.

SystemRequirements MOSEK and MOSEK License

LinkingTo Matrix

Depends R(>= 2.10.0), Matrix(>= 0.9996875-3)

License LGPL (>= 2.1)

URL http://rmosek.r-forge.r-project.org/, http://www.mosek.com/

Repository CRAN

Repository/R-Forge/Project rmosek

Repository/R-Forge/Revision 120

Repository/R-Forge/DateTimeStamp 2013-03-12 10:24:35

Date/Publication 2014-12-13 17:23:24

NeedsCompilation yes

R topics documented:

`mosek` *Solve an optimization problem*

Description

Solve an optimization problem using the MOSEK Optimization Library.

Please see the 'userguide.pdf' for a detailed introduction to this package. This file is located in the "doc" directory at the root of this package:
`system.file("doc", "userguide.pdf", package="Rmosek")`

Usage

```
mosek(problem, opts = list())
```

Arguments

`problem` The optimization problem.

problem	LIST	
..$sense	STRING	
..$c	NUMERIC VECTOR	
..$c0	NUMERIC	(OPTIONAL)
..$A	SPARSE MATRIX	
..$bc	NUMERIC MATRIX (2 rows)	
..$bx	NUMERIC MATRIX (2 rows)	
..$cones	LIST MATRIX (2 rows)	(OPTIONAL)
..$intsub	NUMERIC VECTOR	(OPTIONAL)
..$qobj	LIST	(OPTIONAL)
..$scopt	LIST	(OPTIONAL)
..$iparam/$dparam/$sparam	LIST	(OPTIONAL)
....$<MSK_PARAM>	STRING / NUMERIC	(OPTIONAL)
..$sol	LIST	(OPTIONAL)
....$itr/$bas/$int	LIST	(OPTIONAL)

`opts` The interface options.

opts	LIST	(OPTIONAL)
..$verbose	NUMERIC	(OPTIONAL)
..$usesol	BOOLEAN	(OPTIONAL)
..$useparam	BOOLEAN	(OPTIONAL)
..$soldetail	NUMERIC	(OPTIONAL)
..$getinfo	BOOLEAN	(OPTIONAL)
..$writebefore	STRING (filepath)	(OPTIONAL)
..$writeafter	STRING (filepath)	(OPTIONAL)

Details

The optimization problem should be described in a named list of definitions. The number of variables in the problem is determined from the number of columns in the constraint matrix `A`.

Like a Linear Program it has a linear objective with one coefficient in `c` for each variable, some optional constant `c0`, and the improving direction `sense`. Quadratic terms can be added to the objective with `qobj`. The constraints can either be linear, specified as rows in `A` with lower and upper bounds as columns in `bc` (you can use `Inf` if needed), or conic as specified in the list-typed matrix `cones` (add constraints copyx=x if some variable `x` appears in multiple cones). All variables have lower and upper bounds as columns in `bx`, and will be integer if they appear in the `intsub` list.

As an advanced feature, non-linear unary operators involving exponential or logarithmic functions can be added with `scopt`. Parameters can also be specified for the MOSEK call. `iparam` is integer-typed parameters, `dparam` ia double-typed parameters and `sparam` is string-typed parameters. These parameters can be ignored by setting the option `useparam` to FALSE (the default is TRUE).

Initial solutions are specified in `sol` and should have the same format as the solution returned by the function call. This solution can be ignored by setting the option `usesol` to FALSE (the default is TRUE).

The amount of information printed by the interface can be limited by `verbose` (default=10). The generated model can be exported to any standard modeling fileformat (e.g. lp, opf, lp or mbt), with (resp. without) the identified solution using `writeafter` (resp. `writebefore`).

The optimization process can be terminated at any moment using CTRL + C.

problem	Problem description
.$sense	Objective sense, e.g. "max" or "min"
.$c	Objective coefficients
.$c0	Objective constant
.$A	Constraint matrix
.$bc	Lower and upper constraint bounds
.$bx	Lower and upper variable bounds
.$qobj	Quadratic objective terms
.$cones	Conic constraints
.$intsub	Integer variable indexes
.$scopt	Separable convex optimization
.$iparam/$dparam/$sparam	Parameter list
..$<MSK_PARAM>	Value of any <MSK_PARAM>
.$sol	Initial solution list
..$itr/$bas/$int	Initial solution description
opts	Options
.$verbose	Output logging verbosity
.$usesol	Whether to use the initial solution
.$useparam	Whether to use the specified parameter settings
.$soldetail	Level of detail used to describe solutions.
.$getinfo	Whether to extract MOSEK information items
.$writebefore	Filepath used to export model
.$writeafter	Filepath used to export model and solution

Value

`r` The returned results.

r	LIST	
..$response	LIST	
....$code	NUMERIC	
....$msg	STRING	
..$sol	LIST	
....$itr/$bas/$int	LIST	(SOLVER DEPENDENT)
......$solsta	STRING	
......$prosta	STRING	
......$skc	STRING VECTOR	
......$skx	STRING VECTOR	
......$skn	STRING VECTOR	(NOT IN $bas)
......$xc	NUMERIC VECTOR	
......$xx	NUMERIC VECTOR	
......$slc	NUMERIC VECTOR	(NOT IN $int)
......$suc	NUMERIC VECTOR	(NOT IN $int)
......$slx	NUMERIC VECTOR	(NOT IN $int)
......$sux	NUMERIC VECTOR	(NOT IN $int)
......$snx	NUMERIC VECTOR	(NOT IN $int/$bas)
......$pobjval	NUMERIC	*
......$dobjval	NUMERIC	*(NOT IN $int)
......$pobjbound	NUMERIC	*($int ONLY)
......$maxinfeas	LIST	*
........$pbound	NUMERIC	*
........$peq	NUMERIC	*
........$pcone	NUMERIC	*(NOT IN $bas)
........$dbound	NUMERIC	*(NOT IN $int)
........$deq	NUMERIC	*(NOT IN $int)
........$dcone	NUMERIC	*(NOT IN $int/$bas)
........$int	NUMERIC	*($int ONLY)
..$iinfo/$dinfo	LIST	*
....$<MSK_INFO>	NUMERIC	*

*Starred items must be requested using an option.

The result is a named list containing the response of the MOSEK optimization library. A response code of zero is the signal of success.

Depending on the specified solver, one or more solutions may be returned. The interior-point solution `itr`, the basic (corner point) solution `bas`, and the integer solution `int`.

The problem status `prosta` in all solutions shows the feasibility of your problem description. All solutions are described by a solution status `solsta` (e.g. optimal) along with the variable and constraint activities. All activities will further have a bound key that specify their value in relation

to the declared bounds.

Dual variables are returned for all defined bounds wherever possible. Integer solutions `int` does not have any dual variables as such definitions would not make sense. Basic (corner point) solutions `bas` would never be returned if the problem had conic constraints, and does not define `snx`.

Setting option `soldetail` larger than 1 extracts `pobjval`, `pobjval` and `pobjbound`. Larger than 2 extracts `maxinfeas`. Setting option `getinfo` to `TRUE` extracts `iinfo` and `dinfo`.

r	Result
.$response	Response from the MOSEK Optimization Library
..$code	ID-code of response
..$msg	Human-readable message
.$sol	All solutions identified
..$itr/$bas/$int	Solution description
...$solsta	Solution status
...$prosta	Problem status
...$skc	Linear constraint status keys
...$skx	Variable bound status keys
...$skn	Conic constraint status keys
...$xc	Constraint activities
...$xx	Variable activities
...$slc	Dual variable for constraint lower bounds
...$suc	Dual variable for constraint upper bounds
...$slx	Dual variable for variable lower bounds
...$sux	Dual variable for variable lower bounds
...$snx	Dual variable of conic constraints
...$pobjval	Primal objective value
...$dobjval	Dual objective value
...$pobjbound	Best primal objective bound from relaxations
...$maxinfeas	Maximal solution infeasibilities
....$pbound	Primal inequality constraints
....$peq	Primal equality constraints
....$pcone	Primal cone constraints
....$dbound	Dual inequality constraints
....$deq	Dual equality constraints
....$dcone	Dual cone constraints
....$int	Integer variables
.$iinfo/$dinfo	MOSEK information list
..$<MSK_INFO>	Value of any <MSK_INFO>

See Also

`mosek_version` `mosek_clean`

Examples

```
lo1 <- list()
lo1$sense <- "max"
lo1$c <- c(3,1,5,1)
```

```
lo1$A <- Matrix(c(3,1,2,0,
                  2,1,3,1,
                  0,2,0,3), nrow=3, byrow=TRUE, sparse=TRUE)
lo1$bc <- rbind(blc = c(30,15,-Inf),
                buc = c(30,Inf,25))
lo1$bx <- rbind(blx = c(0,0,0,0),
                bux = c(Inf,10,Inf,Inf))
r <- mosek(lo1, list( soldetail = 1 ))
```

`mosek_clean` *Release an acquired MOSEK license*

Description

Forces the early release of any previously acquired MOSEK license. If you do not share a limited number of licenses among multiple users, you do not need to use this function. Notice that the acquisition of a new MOSEK license will automatically take place at the next call to the function `mosek` given a valid problem description, using a small amount of extra time.

For advanced users: If you utilize the .Call convention directly, bypassing the mosek R-function definition, an Rf_error will result in an unclean memory space. For this reason you can also use `mosek_clean` to tidy up in uncleaned resources after an error has occurred. Otherwise this will not happen until the next call to `mosek` or until the library is unloaded.

Usage

```
mosek_clean()
```

See Also

`mosek`

`mosek_lptoprob` *Construct problem from a linear program*

Description

Construct a problem description from the following linear program:

minimize:	f'x
subject to:	
	A x <= b
	Aeq x = beq
with bounds:	
	lb <= x <= ub

The result of this function is compatible with the problem description of the mosek function.

Usage

```
mosek_lptoprob(f,A,b,Aeq,beq,lb,ub)
```

Arguments

f	Objective coefficients (size n)
A	Constraint inequality matrix (size mA x n)
b	Constraint inequality upper bounds (size mA)
Aeq	Constraint equality matrix (size mEQ x n)
beq	Constraint equality fixed values (size mEQ)
lb	Variable lower bounds (size n)
ub	Variable upper bounds (size n)

See Also

mosek mosek_qptoprob

Examples

```
# Define a linear program
f <- c(0,-5,0)
A <- Matrix(c( 4, 3, 0,
              -2,-1, 0,
               0, 2,-1), nrow=3, byrow=TRUE, sparse=TRUE)
b <- c(8,-2,0)
Aeq <- NA;
beq <- NA;
lb <- rep(-Inf, 3);
ub <- rep(Inf, 3);

# Construct and solve problem
prob <- mosek_lptoprob(f, A, b, Aeq, beq, lb, ub);
r <- mosek(prob);

# Objective value is
print(prob$c %*% r$sol$bas$xx);
```

`mosek_qptoprob` *Construct problem from a quadratic program*

Description

Construct a conic problem description from the following quadratic program:

minimize:	f'x + 0.5x'(F'F)x
subject to:	
	A x <= b
	Aeq x = beq
with bounds:	
	lb <= x <= ub

Given that F is not known, but Q = F'F on the other hand is, we can estimate F from Q by the Cholesky Decomposition: F = Matrix::chol(Q). The result of the `mosek_qptoprob` function is compatible with the problem description of the `mosek` function.

Note that problems with a quadratic objective can also be formulated without cones, using the field 'qobj' in the problem description. This is documented in the userguide.

Usage

```
mosek_qptoprob(F,f,A,b,Aeq,beq,lb,ub)
```

Arguments

`F`	Objective quadratic coefficient matrix (size mF x n)
`f`	Objective linear coefficient vector (size n)
`A`	Constraint inequality matrix (size mA x n)
`b`	Constraint inequality upper bounds (size mA)
`Aeq`	Constraint equality matrix (size mEQ x n)
`beq`	Constraint equality fixed values (size mEQ)
`lb`	Variable lower bounds (size n)
`ub`	Variable upper bounds (size n)

See Also

`mosek` `mosek_lptoprob`

Examples

```
# Define a quadratic program
F <- Diagonal(3)
f <- c(0,-5,0)
A <- Matrix(c( 4, 3, 0,
```

```
              -2,-1, 0,
               0, 2,-1), nrow=3, byrow=TRUE, sparse=TRUE)
b <- c(8,-2,0)
Aeq <- NA;
beq <- NA;
lb <- rep(-Inf, 3);
ub <- rep(Inf, 3);

# Construct and solve problem
prob <- mosek_qptoprob(F, f, A, b, Aeq, beq, lb, ub);
r <- mosek(prob);

# Objective value is
print(prob$c %*% r$sol$itr$xx);
```

`mosek_read` *Read problem from a model file*

Description

Interprets a model from any standard modeling fileformat (e.g. lp, opf, mps, mbt, etc.), controlled by a set of options. The result contains an optimization problem which is compliant with the input specifications of function `mosek`.

Usage

```
mosek_read(modelfile, opts = list())
```

Arguments

`modelfile` The file containing an optimization model.

modelfile STRING (filepath)

`opts` The interface options.

opts	LIST	(OPTIONAL)
..$verbose	NUMERIC	(OPTIONAL)
..$usesol	BOOLEAN	(OPTIONAL)
..$useparam	BOOLEAN	(OPTIONAL)
..$getinfo	BOOLEAN	(OPTIONAL)
..$scofile	STRING (filepath)	(OPTIONAL)
..$matrixformat	STRING	(OPTIONAL)

Details

The `modelfile` should be an absolute or relative path to a model file.

The amount of information printed by the interface can be limited by `verbose` (default=10). Whether to read the initial solution, if one such exists in the model file, is indicated by `usesol` which by default is FALSE. Whether to read the full list of parameter settings, some of which may have been defined by the model file, is indicated by `useparam` which by default is FALSE.

The option `scofile` is used in separable convex optimization to specify the absolute or relative path to the operator file.

The format of the imported constraint matrix is controlled by `matrixformat` and can be either sparse coordinate `COO`, compressed sparse column `CSC`, or a list-based alternative `simple:COO`. The matrix formats `CSC` and `COO` are based on the package 'Matrix' superclasses `CsparseMatrix` and `TsparseMatrix`.

modelfile	Filepath to the model
opts	Options
.$verbose	Output logging verbosity
.$usesol	Whether to read an initial solution
.$useparam	Whether to read all parameter settings
.$getinfo	Whether to extract MOSEK information items
.$scofile	Source of operators read to scopt
.$matrixformat	The sparse format of the constraint matrix

Value

`r` The returned result.

r	LIST	
..$response	LIST	
....$code	NUMERIC	
....$msg	STRING	
..$prob	LIST	
..$iinfo/$dinfo	LIST	*
....$<MSK_INFO>	NUMERIC	*

*Starred items must be requested using an option.

The result is a named list containing the response of the MOSEK Optimization Library when reading the model file. A response code of zero is the signal of success.

On success, the result contains the problem specification with all problem data. This problem specification is compliant with the input specifications of function `mosek`.

Setting option `getinfo` to `TRUE` extracts `iinfo` and `dinfo`.

r Result

.$response	Response from the MOSEK Optimization Library
..$code	ID-code of response
..$msg	Human-readable message
.$prob	Problem description
.$iinfo/$dinfo	MOSEK information list
..$<MSK_INFO>	Value of any <MSK_INFO>

See Also

mosek mosek_write

Examples

```
modelfile <- system.file(package="Rmosek", "extdata", "lo1.opf")
rr <- mosek_read(modelfile)
if (!identical(rr$response$code, 0))
  stop("Failed to read model file")
rlo1 <- mosek(rr$prob)

modelfile <- system.file(package="Rmosek", "extdata", "milo1.opf")
rr <- mosek_read(modelfile)
if (!identical(rr$response$code, 0))
  stop("Failed to read model file")
rmilo1 <- mosek(rr$prob)

modelfile <- system.file(package="Rmosek", "extdata", "cqo1.opf")
rr <- mosek_read(modelfile)
if (!identical(rr$response$code, 0))
  stop("Failed to read model file")
rcqo1 <- mosek(rr$prob)

modelfile <- system.file(package="Rmosek", "extdata", "qo1.opf")
rr <- mosek_read(modelfile)
if (!identical(rr$response$code, 0))
  stop("Failed to read model file")
rqo1 <- mosek(rr$prob)

modelfile <- system.file(package="Rmosek", "extdata", "sco1.opf")
modelscofile <- system.file(package="Rmosek", "extdata", "sco1.sco")
rr <- mosek_read(modelfile, list(scofile=modelscofile))
if (!identical(rr$response$code, 0))
  stop("Failed to read model file")
rsco1 <- mosek(rr$prob)
```

---

mosek_version *Version of the MOSEK Optimization Library*

Description

Retrieves a string containing the version number of the utilized MOSEK Optimization Library.

Usage

```
mosek_version()
```

See Also

mosek

`mosek_write` *Write problem to a model file*

Description

Outputs a model of an optimization problem in any standard modeling fileformat (e.g. lp, opf, mps, mbt, etc.), controlled by a set of options. The modeling fileformat is selected based on the extension of the modelfile.

Usage

```
mosek_write(problem, modelfile, opts = list())
```

Arguments

`problem` The optimization problem.

problem LIST

`modelfile` The file to write the optimization model.

modelfile STRING (filepath)

`opts` The interface options.

opts	LIST	(OPTIONAL)
..$verbose	NUMERIC	(OPTIONAL)
..$usesol	BOOLEAN	(OPTIONAL)
..$useparam	BOOLEAN	(OPTIONAL)
..$getinfo	BOOLEAN	(OPTIONAL)
..$scofile	STRING (filepath)	(MANDATORY IN SCOPT)

Details

The `problem` should be compliant with the input specification of function `mosek`. Please see this function for more details.

The `modelfile` should be an absolute or relative path to the model file. If the file extension is `.opf`, the model will be written in the Optimization Problem Format. Other formats include lp, mps and mbt.

The amount of information printed by the interface can be limited by `verbose` (default=10). Whether to write the initial solution, if one such exists in the problem description, is indicated by `usesol` which by default is FALSE. Whether to write the full list of parameter settings, some of which may have been specified by the problem description, is indicated by `useparam` which by default is FALSE.

The option `scofile` is used in separable convex optimization to specify the absolute or relative path to the operator file.

problem	Problem description
.$modelfile	Filepath to the model
.$opts	Options
..$verbose	Output logging verbosity
..$usesol	Whether to write an initial solution
..$useparam	Whether to write all parameter settings
..$getinfo	Whether to extract MOSEK information items
..$scofile	Destination of operators from scopt

Value

`r` The returned result.

r	LIST	
..$response	LIST	
....$code	NUMERIC	
....$msg	STRING	
..$iinfo/$dinfo	LIST	*
....$<MSK_INFO>	NUMERIC	*

*Starred items must be requested using an option.

The result is a named list containing the response of the MOSEK Optimization Library when writing to the model file. A response code of zero is the signal of success.

Setting option `getinfo` to `TRUE` extracts `iinfo` and `dinfo`.

r	Result
.$response	Response from the MOSEK Optimization Library
..$code	ID-code of response

..$msg	Human-readable message
.$iinfo/$dinfo	MOSEK information list
..$<MSK_INFO>	Value of any <MSK_INFO>

See Also

mosek mosek_read

Examples

```
lo1 <- list()
lo1$sense <- "max"
lo1$c <- c(3,1,5,1)
lo1$A <- Matrix(c(3,1,2,0,
                  2,1,3,1,
                  0,2,0,3), nrow=3, byrow=TRUE, sparse=TRUE)
lo1$bc <- rbind(blc = c(30,15,-Inf),
                buc = c(30,Inf,25));
lo1$bx <- rbind(blx = c(0,0,0,0),
                bux = c(Inf,10,Inf,Inf));
rr <- mosek_write(lo1, "lo1.opf")
if (!identical(rr$response$code, 0))
  stop("Failed to write model file to current working directory")
```

Index